

$P = NP$

Sergey V. Yakhontov
 Ph.D. in Computer Science
 Faculty of Mathematics and Mechanics
 Saint Petersburg State University
 Russian Federation
 e-mail: SergeyV.Yakhontov@gmail.com
 phone: +7-911-966-84-30
 25-Sept-2012

Abstract

The present work proves that $P=NP$. The proof, presented in this work, is a constructive one: the program of polynomial time deterministic multi-tape Turing machine $M_{\exists AcceptPaths}$, that determines if there exist polynomial-length accepting computation paths of non-deterministic single-tape Turing machine M_{NP} , is constructed.

Machine $M_{\exists AcceptPaths}$ does not count all the accepting computation paths of machine M_{NP} (in this case there would be class $\#P$), but machine $M_{\exists AcceptPaths}$ only determines if there exists at least one polynomial-length accepting computation path of machine M_{NP} .

Keywords: Computational complexity, Turing machine, class P , class NP , the P vs NP Problem, accepting computation paths.

1. Introduction. This work concerns the complexity classes of languages over a finite alphabet that are decidable by Turing machines. It follows from the definition of classes P and NP [1, 2] that $P \subseteq NP$ wherein P is the shortened indication of P_{TIME} and NP is the shortened indication of NP_{TIME} . However, the problem of strictness of the inclusion, referred to as **the P versus NP Problem**, is one of the most important unsolved problems in the theory of computational complexity.

The P versus NP Problem was introduced by Stephen Cook in 1971 [3] and independently by Leonid Levin in 1973 [4]. A detailed description of the problem in [5] formulates it as follows: can each language over a finite alphabet (wherein the number of symbols is equal to or more than two), which is decidable by **a polynomial time non-deterministic single-tape Turing machine**, also be decided by **a polynomial time deterministic single-tape Turing machine**?

The present work proves that $P=NP$. The proof, presented in this work, is a constructive one: the program of polynomial time deterministic multi-tape Turing machine $M_{\exists AcceptPaths}$, that determines if there exist polynomial-length accepting computation paths of a non-deterministic single-tape Turing machine M_{NP} , is constructed.

The concept of the construction of machine $M_{\exists AcceptPaths}$ is to subtract the count of the tape-inconsistent sequences with length μ of local configurations of machine M_{NP} from the count of the tape-arbitrary sequences with length μ of local configurations of machine M_{NP} to get the count of the tape-consistent sequences with length μ of local configurations of machine M_{NP} . The count of the tape-consistent sequences with length μ of local configurations of machine M_{NP} is equal to the count of the accepting computation paths with length μ of machine M_{NP} (local configurations and sequences of local configurations are defined below).

The space used to calculate the local configurations of a tape-inconsistent sequence or a tape-arbitrary sequence of local configurations of polynomial time non-deterministic single-tape Turing machine M_{NP} is logarithmic only, so the deterministic Turing machines, that are used in this work to count such sequences of local configurations of machine M_{NP} , work in polynomial time.

Machine $M_{\exists \text{AcceptPaths}}$ does not count all the accepting computation paths of machine M_{NP} (in this case there would be complexity class $\#P$), but machine $M_{\exists \text{AcceptPaths}}$ only determines if there exists at least one polynomial-length accepting computation path of machine M_{NP} .

Most of the works on the **P** versus **NP** Problem could be found on the Internet at [6] and [7].

2. Non-deterministic computations. Let $M = \langle Q, \Gamma, b, \Sigma, \Delta, q_{start}, F \rangle$ be a non-deterministic single-tape Turing machine wherein Q is the set of states, Γ is the set of tape symbols, b is the blank symbol, Σ is the set of input symbols, Δ is the transition relation, q_{start} is the initial state and F is the set of accepting states. The elements of the set $\{L, R, S\}$ denote, as is usual, the moves of the tape head of machine M .

Non-deterministic Turing machines as decision procedures are usually defined as follows.

Definition 1. *Non-deterministic Turing machine M accepts input x if all the computation paths of machine M on input x are accepting computation paths.*

Definition 2. *Non-deterministic Turing machine M rejects input x if all the computation paths of machine M on input x are finite and these paths are not accepting computation paths.*

Definition 3. *Non-deterministic Turing machine M decides language $A \in \Sigma^*$ if machine M accepts each $x \in A$ and rejects each $x \notin A$.*

The time (space) computational complexity of non-deterministic Turing machine M is polynomial if there exists a polynomial $t(n)$ ($s(n)$ accordingly) such that for any input x

- 1) the minimum of the lengths of all the accepting computation paths of machine M on input x does not exceed $t(|x|)$ (accordingly, the number of the different visited cells on each computation path does not exceed $s(|x|)$) if machine M accepts input x , and
- 2) the lengths of all the computation paths of machine M on input x do not exceed $t(|x|)$ (accordingly, the number of the different visited cells on each computation path does not exceed $s(|x|)$) if machine M rejects input x .

Here, (as is usual) by means of $|x|$ the length of word x is specified.

Definition 4. *Let μ be an integer. Computation path P of machine M on input x is said to be μ -length computation path if the length of P is equal to μ . Accepting computation path P of machine M on input x is said to be μ -length accepting computation path if P is μ -length computation path.*

If Turing machine M accepts input x and the time complexity of machine M is bounded above by polynomial $p(n)$ then the computation tree of machine M on input x has at least one μ -length accepting computation path where $\mu \leq p(|x|)$.

If Turing machine M rejects input x and the time complexity of machine M is bounded above by polynomial $p(n)$ then all the computation paths of machine M on input x are precisely the μ -length computation paths, wherein $\mu \leq p(|x|)$ (μ can be different for different paths), and these paths are not accepting computation paths.

3. The notion of sequences of local configurations. The following definitions are used to construct Turing machine $M_{\exists \text{AcceptPaths}}$.

Definition 5. *Local configuration of machine M is defined to be tuple $(q, s, q', s', m, \kappa^{tape}, \kappa^{step})$ wherein $q, q' \in Q$, $s, s' \in \Gamma$, $m \in \{L, R, S\}$ and $\kappa^{tape}, \kappa^{step}$ are integers.*

Definition 6. Local configuration $t = (q, s, q', s', m, \kappa^{tape}, \kappa^{step})$ is said to be a tape-consistent local configuration if $d = ((q, s), (q', s', m)) \in \Delta$.

Definition 7. Local configuration $t = (q, s, q', s', m, \kappa^{tape}, \kappa^{step})$ is said to be a tape-inconsistent local configuration if $d = ((q, s''), (q', s', m)) \in \Delta$ and $s'' \neq s$.

Notation 1. We write $d \triangle t$ in the case of $d = ((q, s), (q', s', m)) \in \Delta$, and we write $d \nabla t$ in the case of $d = ((q, s''), (q', s', m)) \in \Delta$ and $s'' \neq s$.

Definition 8. Tape-arbitrary local configuration is defined to be a tape-consistent or tape-inconsistent local configuration.

Definition 9. Let $t_1 = (q_1, s_1, q'_1, s'_1, m_1, \kappa_1^{tape}, \kappa_1^{step})$ and $t_2 = (q_2, s_2, q'_2, s'_2, m_2, \kappa_2^{tape}, \kappa_2^{step})$ be local configurations. Pair (t_1, t_2) is said to be sequential pair if $q_2 = q'_1$, $\kappa_2^{step} = \kappa_1^{step} + 1$ and

- 1) if $m_1 = L$ then $\kappa_2^{tape} = \kappa_1^{tape} - 1$;
- 2) if $m_1 = R$ then $\kappa_2^{tape} = \kappa_1^{tape} + 1$;
- 3) if $m_1 = S$ then $\kappa_2^{tape} = \kappa_1^{tape}$.

We consider only finite sequences of local configurations such that each pair of local configurations is a sequential pair.

Definition 10. Pair of local configurations

$$\begin{aligned} t_{i_1} &= (q_{i_1}, s_{i_1}, q'_{i_1}, s'_{i_1}, m_{i_1}, \kappa_{i_1}^{tape}, \kappa_{i_1}^{step}) \text{ and} \\ t_{i_2} &= (q_{i_2}, s_{i_2}, q'_{i_2}, s'_{i_2}, m_{i_2}, \kappa_{i_2}^{tape}, \kappa_{i_2}^{step}) \end{aligned}$$

is said to be a tape-consistent pair of local configurations if $s_{i_2} = s'_{i_1}$. Otherwise (when $s_{i_2} \neq s'_{i_1}$) the pair is said to be a tape-inconsistent pair of local configurations.

Definition 11. $\kappa = \min\{j | t = (q, s, q', s', m, \kappa^{tape}, j) \in L\}$, wherein L is a sequence of local configurations, is denoted by $\text{TapeFirst}[L, \kappa^{tape}]$.

Definition 12. $\kappa = \max\{j | j < \kappa^{step} \text{ \& } t = (q, s, q', s', m, \kappa^{tape}, j) \in L\}$, wherein L is a sequence of local configurations, is denoted by $\text{TapePrev}[L, \kappa^{tape}, \kappa^{step}]$.

Definition 13. Sequence $L_{sub} = (t_1, \dots, t_\mu)$ of local configurations, denoted by $\text{Subseq}[L, \kappa^{tape}]$, is said to be a subsequence of sequence L of local configurations at cell κ if $\kappa^{tape} = \kappa$ for each local configuration $t = (q, s, q', s', m, \kappa^{tape}, \kappa^{step})$ in L_{sub} .

Let's enumerate the tape cells of Turing machine M on input x as follows: the number of the cell c_1 , containing the leftmost symbol of input x , is 1, the number of the cell right to c_1 is 2, the number of the cell left to c_1 is 0, and so on.

Notation 2. Let x be an input of machine M . The symbol in the tape cell κ is denoted by $\text{Tape}[x, \kappa]$.

Notation 3. Integer range $[1 - (\mu - 1) .. (|x| + (\mu - 1))]$ is denoted by $\text{TapeRange}[x, \mu]$.

Let's note that $\kappa^{tape} \in \text{TapeRange}[x, \mu]$ for each local configuration $t = (q, s, q', s', m, \kappa^{tape}, \kappa^{step})$ in μ -length sequence L of local configurations.

Definition 14. We say that sequence $L = (t_1, \dots, t_\mu)$ of local configurations starts on input x if $t_1 = (q_{start}, s, q', s', m, 1, 1)$ for some s, q', s', m .

Definition 15. We say that sequence $L = (t_1, \dots, t_\mu)$ of local configurations corresponds to input x at cell κ^{tape} if one of the following holds:

- 1) there exists $t = (q, s, q', s', m, \kappa^{tape}, \kappa^{step}) \in \text{Subseq}[L, \kappa^{tape}]$ wherein $s = \text{Tape}[x, \kappa^{tape}]$ and $\kappa^{step} = \text{TapeFirst}[L, \kappa^{tape}]$;
- 2) there is no $t = (q, s, q', s', m, \kappa^{tape}, \kappa^{step}) \in \text{Subseq}[L, \kappa^{tape}]$.

Definition 16. Sequence $L = (t_1, \dots, t_\mu)$ of local configurations of machine M on input x is said to be a tape-consistent sequence of local configurations on input x if the following holds:

- 1) L starts on input x ;
- 2) L corresponds to input x at each cell $\kappa \in \text{TapeRange}[x, \mu]$;
- 3) each t_i is a tape-consistent local configuration;
- 4) for each $\kappa \in \text{TapeRange}[x, \mu]$ the following holds: each pair (t_i, t_{i+1}) in $\text{Subseq}[L, \kappa]$ is a tape-consistent pair of local configurations.

Definition 17. Sequence $L = (t_1, \dots, t_\mu)$ of local configurations of machine M on input x is said to be a tape-inconsistent at pair $(\kappa^{\text{tape}}, \kappa^{\text{step}})$ sequence of local configurations on input x if the following holds:

- 1) $t = (q, s, q', s', m, \kappa^{\text{tape}}, \kappa^{\text{step}}) \in L$;
- 2) L starts on input x ;
- 3) each $t_i, i \neq \kappa^{\text{step}}$, is a tape-arbitrary local configuration;
- 4) one of the following holds:
 - 4.1) $t_{\kappa^{\text{step}}}$ is a tape-inconsistent local configuration;
 - 4.2) if $\kappa^{\text{step}} = \text{TapeFirst}[L, \kappa^{\text{tape}}]$ then $s \neq \text{Tape}[x, \kappa^{\text{tape}}]$;
 - 4.3) if $\kappa = \text{TapePrev}[L, \kappa^{\text{tape}}, \kappa^{\text{step}}]$ then pair $(t_{\kappa^{\text{tape}}}, t_\kappa)$ is a tape-inconsistent pair of local configurations.

Definition 18. Sequence $L = (t_1, \dots, t_\mu)$ of local configurations of machine M on input x is said to be a tape-inconsistent sequence of local configuration on input x if L is tape-inconsistent at some pair $(\kappa^{\text{tape}}, \kappa^{\text{step}})$ sequence on input x .

Definition 19. Sequence $L = (t_1, \dots, t_\mu)$ of local configurations of machine M on input x is said to be tape-arbitrary sequence of local configurations if the following holds:

- 1) L starts on input x ;
- 2) each t_i is a tape-arbitrary local configuration;

Definition 20. Sequence $L = (t_1, \dots, t_\mu)$ of local configurations of machine M is said to be q' -sequence of local configurations if $t_\mu = (q, s, q', s', m, \kappa^{\text{tape}}, \kappa^{\text{step}})$.

Definition 21. Sequence $L = (t_1, \dots, t_\mu)$ of local configurations of machine M is said to be an accepting sequence if $t_\mu = (q, s, q', s', m, \kappa^{\text{tape}}, \kappa^{\text{step}})$ wherein $q' \in F$.

Definition 22. Sequence $L = (t_1, \dots, t_\mu)$ of local configurations of machine M is said to be a rejecting sequence if there is no transition in Δ from configuration $t_\mu = (q, s, q', s', m, \kappa^{\text{tape}}, \kappa^{\text{step}})$ wherein $q' \notin F$.

Definition 23. Sequence $L = (t_1, \dots, t_\mu)$ of local configurations of machine M is said to be μ -length sequence of local configurations.

Definition 24. Tape-consistent sequence $L = (t_1, \dots, t_\mu)$ of local configurations is said to be the sequence corresponding to computation path $P = \alpha_1 \dots \alpha_{\mu-1}$ of machine M on input x if each d_i , such that $d_i \triangle t_i$, is the transition corresponding to configuration transition $\alpha_i \vdash \alpha_{i+1}$.

Definition 25. Tree T of local configurations is said to be a tape-consistent tree of local configurations if each path of T from the root to a leaf is a tape-consistent sequence of local configurations.

In a similar way, tape-inconsistent at step κ tree, tape-inconsistent tree, tape-arbitrary tree, q -tree, accepting tree, rejecting tree, μ -length tree of local configurations are defined.

4. The concept of the construction of Turing machine $M_{\exists \text{AcceptPaths}}$. The concept of the construction of Turing machine $M_{\exists \text{AcceptPaths}}$ is based on the following lemma.

Lemma 1. *There is a one-to-one mapping from the set of μ -length tape-consistent sequences of local configurations of machine M on input x onto the set of μ -length sequences of local configurations of machine M on input x that correspond to the $(\mu - 1)$ -length computation paths of machine M on input x .*

Proof. The lemma follows directly from the definition. □

To count the tape-consistent sequences of local configurations, we should keep all the symbols written onto the tape of machine M ; therefore, we use polynomial space.

If we count the tape-inconsistent sequences of local configurations or count the tape-arbitrary sequences of local configurations, we should know the current local configuration and the symbols in one of the tape cells; therefore, we use logarithmic space.

Notation 4. *Let $\text{ConsistSet}[x, q, \mu]$ be the set of μ -length tape-consistent q -sequences of local configurations of machine M on input x . Let $\text{ConsistCount}[x, q, \mu]$ be $|\text{ConsistSet}[x, q, \mu]|$.*

Here, (as is usual) by means of $|S|$ the cardinality of set S is specified.

Notation 5. *Let $\text{InconsistSet}[x, q, \kappa^{\text{tape}}, \kappa^{\text{step}}, \mu]$ be the set of μ -length tape-inconsistent at pair $(\kappa^{\text{tape}}, \kappa^{\text{step}})$ q -sequences of local configurations of machine M on input x . Let $\text{InconsistCount}[x, q, \kappa^{\text{tape}}, \kappa^{\text{step}}, \mu]$ be $|\text{InconsistSet}[x, q, \kappa^{\text{tape}}, \kappa^{\text{step}}, \mu]|$.*

Notation 6. *Let $\text{InconsistSet}[x, q, \mu]$ be the set of μ -length tape-inconsistent q -sequences of local configurations of machine M on input x . Let $\text{InconsistCount}[x, q, \mu]$ be $|\text{InconsistSet}[x, q, \mu]|$.*

Notation 7. *Let $\text{ArbitrarySet}[x, q, \mu]$ be the set of μ -length tape-arbitrary q -sequences of local configurations of machine M on input x . Let $\text{ArbitraryCount}[x, q, \mu]$ be $|\text{ArbitrarySet}[x, q, \mu]|$.*

Notation 8. $Q[\text{NotAccept}] = \{q | q \notin F\}$, $Q[\text{Any}] = Q$.

Definition 26. $\text{ConsistCount}[x, S, \mu] = \cup_{q \in S} \text{ConsistCount}[x, q, \mu]$ for some set S of the states of machine M .

Similar definitions are introduced for tape-inconsistent and tape-arbitrary sequences of local configurations.

Lemma 2.

$$\text{ConsistCount}[x, q, \mu] = \text{ArbitraryCount}[x, q, \mu] - \text{InconsistCount}[x, q, \mu].$$

Proof. We should show that

$$\begin{aligned} \text{ConsistSet}[x, q, \mu] \cap \text{InconsistSet}[x, q, \mu] &= \emptyset \text{ and} \\ \text{ArbitrarySet}[x, q, \mu] &= \text{ConsistSet}[x, q, \mu] \cup \text{InconsistSet}[x, q, \mu]. \end{aligned}$$

The first equality follows directly from the definition.

Furthermore, inclusions

$$\begin{aligned} \text{ConsistSet}[x, q, \mu] &\subseteq \text{ArbitrarySet}[x, q, \mu] \text{ and} \\ \text{InconsistSet}[x, q, \mu] &\subseteq \text{ArbitrarySet}[x, q, \mu] \end{aligned}$$

follow directly from the definition also.

The rest is to show that $\text{ArbitrarySet}[x, q, \mu] \subseteq \text{ConsistSet}[x, q, \mu] \cup \text{InconsistSet}[x, q, \mu]$.

Let $L = (t_1, \dots, t_\mu)$ be a tape-arbitrary sequence. Then

- 1) if one of 4.1), 4.2) and 4.3) in definition 17 holds for some $t_i \in L$ then $L \in \text{InconsistSet}[x, q, \mu]$;
- 2) otherwise $L \in \text{ConsistSet}[x, q, \mu]$.

To calculate $InconsistCount[x, q, \mu]$, we perform a loop for $i \in TapeRange[x, \mu]$, $j \in 1..\mu$, counting values $InconsistCount[x, q, i, j, \mu]$ at each iteration.

To calculate $InconsistCount[x, q, i, j, \mu]$, it is sufficient to perform deep-first traversal of the tape-inconsistent at pair (i, j) tree $T_{i,j}$ of local configurations on input x , keeping the tree nodes that are reached in table $ConfigTable$. The count of such nodes is polynomial in μ since we use logarithmic space to navigate through the elements of sequences of local configurations.

When traversing tree $T_{i,j}$, we store the count of the subsequences of local configurations, starting from the current local configuration, in table $CountTable$. We also store in table $SubseqTable_{i,j}$ all the subsequences that end with the configurations stored in table $ConfigTable$.

At each step of tree traversing, we fetch the count of the sequences of local configurations from table $CountTable$ and exclude sequences, stored in tables $SeqTable_{i,j}$, from the current set of sequences.

Since the count of local configurations is polynomial in μ , the count of the records in tables $ConfigTable$, $CountTable$ and $SubseqTable_{i,j}$ are polynomial in μ .

In a similar way we calculate $ArbitraryCount[x, q, \mu]$.

Now let M be a non-deterministic single-tape Turing machine that works in time $p(n)$ wherein $p(n)$ is a polynomial. Turing machine $M_{\exists AcceptPaths}$ works as follows: it performs a loop for $\mu \in 1..p(n)$, calculating the following values at each iteration:

$$\begin{aligned} &InconsistCount[x, F, \mu], \text{ ArbitraryCount}[x, F, \mu], \\ &InconsistCount[x, Q[NotAccept], \mu], \text{ ArbitraryCount}[x, Q[NotAccept], \mu], \\ &InconsistCount[x, Q[Any], \mu], \text{ ArbitraryCount}[x, Q[Any], \mu]. \end{aligned}$$

Then, using these values, machine $M_{\exists AcceptPaths}$ calculates the counts of tape-consistent sequences of local configurations:

$$\begin{aligned} &ConsistCount[x, F, \mu] = \text{ArbitraryCount}[x, F, \mu] - \text{InconsistCount}[x, F, \mu], \\ &ConsistCount[x, Q[NotAccept], \mu] = \\ &\quad \text{ArbitraryCount}[x, Q[NotAccept], \mu] - \text{InconsistCount}[x, Q[NotAccept], \mu], \\ &ConsistCount[x, Q[Any], \mu] = \text{ArbitraryCount}[x, Q[Any], \mu] - \text{InconsistCount}[x, Q[Any], \mu]. \end{aligned}$$

Since machine M works in polynomial time $p(n)$, one of the following happens:

- 1) if machine M accepts input x , $|x| = n$, then the loop stops at iteration $\mu \leq p(n)$ such that $ConsistCount[x, F, \mu] > 0$;
- 2) if machine M rejects input x , $|x| = n$, then the loop stops at iteration $\mu \leq p(n)$ such that $ConsistCount[x, Q[NotAccept], \mu] = ConsistCount[x, Q[Any], \mu]$.

5. Non-deterministic multi-tape Turing machines $M_{Inconsist}$ and $M_{Arbitrary}$. Let M_{NP} be a non-deterministic single-tape Turing machine that decides language A and works in time $p(n)$ wherein $p(n)$ is a polynomial.

The input of machine $M_{Inconsist}$ are words $(x, \kappa^{tape}, \kappa^{step}, \mu)$ wherein x is a word and κ^{tape} , κ^{step} and μ are binary integers. All the computation paths of machine $M_{Inconsist}$ on the input x correspond to μ -length tape-inconsistent at pair $(\kappa^{tape}, \kappa^{step})$ sequences of local configurations of machine M_{NP} on input x .

Program of Turing machine $M_{Inconsist}$.

Input: Word $(x, \kappa^{tape}, \kappa^{step}, \mu)$.

- 1) $CellSymbol := Tape[x, \kappa^{tape}]$;

- 2) compute the current configurations $t_k = (q, s, q', s', m, i^{tape}, i^{step})$ of machine M_{NP} taking into account the following:
 - 2.1) if $k = 1$ then $t_1 := (q_{start}, s, q', s', m, 1, 1)$ for some s, q', s, m ;
 - 2.2) calculating t_k , we take into account the symbol stored in variable $CellSymbol$;
 - 2.3) t_k should be such local configuration that one of 4.1), 4.2) and 4.3) in definition 17 holds;
- 3) if $i^{tape} = \kappa^{tape}$ then $CellSymbol := s'$;
- 4) if there is no next local configuration then stop at an rejecting state;
- 5) if the length of the sequence of local configurations being computed is equal to μ then stop at an accepting state.

The program of Turing machine $M_{Arbitrary}$ is similar to the program of machine $M_{Inconsistent}$.

The computation trees of machines $M_{Inconsistent}$ and $M_{Arbitrary}$ correspond to the tape-consistent and tape-arbitrary trees of local configurations of machine M_{NP} on input x .

Lemma 3. *If non-deterministic single-tape Turing machine M_{NP} works in polynomial time then non-deterministic multi-tape Turing machines $M_{Inconsistent}$ and $M_{Arbitrary}$ work in polynomial time and logarithmic space.*

6. Deterministic multi-tape Turing machines $M_{InconsistentCount}$ and $M_{ArbitraryCount}$. Machines $M_{InconsistentCount}$ and $M_{ArbitraryCount}$ perform deep-first traversal of the computation trees of non-deterministic Turing machines $M_{Inconsistent}$ and $M_{Arbitrary}$ to count the sequences of local configurations of machine M_{NP} .

We only take into account the accepting computation paths of machines $M_{Inconsistent}$ and $M_{Arbitrary}$.

Program of Turing machine $M_{InconsistentCount}$.

Input: Word (x, μ) .

Output: $InconsistentCount[x, S, \mu]$.

- 1) $count := 0$;
- 2) for each $i \in TapeRange[x, \mu]$, $j \in 1..\mu$ do the following:
 - 2.1) perform deep-first traversal of the computation tree $T_{i,j}$ of machine $M_{Inconsistent}$ on input (x, i, j, μ) taking into account μ -length q -sequences of local configurations;
 - 2.2) during the deep-first traversal, store the following data in the tables:
 - 2.2.1) store the local configurations, containing in the black nodes, in table $ConfigTable$;
 - 2.2.2) store the counts of the sequences, starting from the current local configuration, in table $CountTable$;
 - 2.2.3) if the current local configuration is contained in a black node then store the subsequence from the root to the current local configuration in table $SubseqTable_{i,j}$;
 - 2.3) during the deep-first traversal, count the sequences starting from the tree nodes:
 - 2.3.1) if the current node has k successors then sum the counts of sequences for all the successors;
 - 2.3.2) if the current local configuration is contained in a black node then fetch the count of sequences from table $CountTable$;
 - 2.3.3) using algorithm $ExcludeSequences$ (defined below), exclude from the count of sequences, taken into account, the count of sequences that are tape-inconsistent at pair (i_{prev}, j_{prev}) for each pair from table $PairTable$;
 - 2.4) fetch the count $\alpha_{i,j}$ of sequences, starting from the root, from table $CountTable$ ($\alpha_{i,j} = InconsistentCount[x, S, i, j, \mu]$);

- 2.5) $count := count + \alpha_{i,j}$;
- 2.6) clear tables *ConfigTable* and *CountTable*;
- 2.7) add pair (i, j) to table *PairTable*;
- 3) write *count* to the output.

Here, (as is usual) black nodes are tree nodes that are visited during deep-first traversal.

Let's note that local configurations contain κ^{step} ; therefore, the program is correct because there are no duplicate local configurations on any computation path.

The program of machine $M_{ArbitraryCount}$ is similar to the program of machine $M_{InconsistentCount}$.

Lemma 4. *Deterministic multi-tape Turing machines $M_{InconsistentCount}$ and $M_{ArbitraryCount}$ work in polynomial time.*

7. Algorithm *ExcludeSequences*. Let $\{T_{i,j} | i \leq m, j \leq n\}$ be a set of computation trees of machine $M_{Inconsistent}$ on inputs (x, i, j, μ) .

Notation 9. Let $P_{i,j}$ be the set of paths in $T_{i,j}$ from the root to the leaves.

Notation 10. Let $G_{i,j}$ be the graph produced in the deep-first traversal of $T_{i,j}$ in machine $M_{InconsistentCount}$.

Notation 11. Let $C_{i,j}$ be the graph of local configurations that are contained in $G_{i,j}$.

The algorithm finds the cardinality of set $P_{m,n} \setminus \bigcup_{i=1..(m-1), j=1..(n-1)} P_{i,j}$.

The algorithm considers graph $C_{m,n}$ as the direct acyclic control flow graph of a program that assigns values to the cell tapes.

The algorithm performs the following steps:

- 1) enumerate all the assignments to the tape cells in the nodes of graph $C_{m,n}$;
- 2) perform reaching definitions analysis on control flow graph $C_{m,n}$;
- 3) using def-use chains pairs, mark nodes of graph $C_{m,n}$, containing tape-inconsistent at steps i, j pairs, with labels lab_p, lab_q ;
- 4) propagate the labels from its original locations in graph $C_{m,n}$ to the leaves;
- 5) propagate the labels from its original locations in graph $C_{m,n}$ to the roots;
- 6) mark all the edges in graph $C_{m,n}$ that contain the pairs (lab_p, lab_q) ;
- 7) using deep-first traversal of graph $C_{m,n}$, count the paths in $P_{m,n}$ excluding the paths containing the marked edges.

Lemma 5. *Algorithm *ExcludeSequences* works in polynomial time.*

8. Deterministic multi-tape Turing machine $M_{\exists AcceptPaths}$. Deterministic multi-tape Turing machine $M_{\exists AcceptPaths}$ is constructed using deterministic Turing machines $M_{InconsistentCount}$ and $M_{ArbitraryCount}$. Machine $M_{\exists AcceptPaths}$ simply subtracts the count of μ -length tape-inconsistent accepting sequences of local configurations from the count of μ -length tape-arbitrary accepting sequences of local configurations to compute the count of μ -length tape-consistent accepting sequences of local configurations that is equal to the count of μ -length accepting computation paths of machine M_{NP} .

Program of Turing machine $M_{\exists AcceptPaths}$.

Input: Word x .

Output: If there exist accepting computation paths of machine M_{NP} on input x .

- 1) $i := 1$;

2) do the following loop for i :

2.1) using machine $M_{InconsistentCount}$, calculate

$$\begin{aligned}\alpha_1 &= InconsistentCount[x, F, i], \\ \alpha_2 &= InconsistentCount[x, Q[NotAccept], i], \\ \alpha_3 &= InconsistentCount[x, Q[Any], i];\end{aligned}$$

2.2) using machine $M_{ArbitraryCount}$, calculate

$$\begin{aligned}\beta_1 &= ArbitraryCount[x, F, i], \\ \beta_2 &= ArbitraryCount[x, Q[NotAccept], i], \\ \beta_3 &= ArbitraryCount[x, Q[Any], i];\end{aligned}$$

2.3) calculate

$$\begin{aligned}\gamma_1 &= \beta_1 - \alpha_1 \text{ } (\gamma_1 \text{ is equal to } ConsistentCount[x, F, i]), \\ \gamma_2 &= \beta_2 - \alpha_2 \text{ } (\gamma_2 \text{ is equal to } ConsistentCount[x, Q[NotAccept], i]), \\ \gamma_3 &= \beta_3 - \alpha_3 \text{ } (\gamma_3 \text{ is equal to } ConsistentCount[x, Q[Any], i]);\end{aligned}$$

2.4) if $\gamma_1 > 0$ then write *Yes* to the output and stop;

2.5) if $\gamma_2 = \gamma_3$ then write *No* to the output and stop.

Lemma 6. *If M_{NP} is a polynomial time non-deterministic single-tape Turing machine, that decides language A , then deterministic multi-tape Turing machine $M_{\exists AcceptPaths}$ determines in polynomial time if there exist polynomial-length accepting computation paths of machine M_{NP} .*

9. Main result. If M is a deterministic multi-tape Turing machine that computes function $f(x)$ and works in time $t(n)$ then we can construct deterministic single-tape Turing machine M' that computes the same function and works in time $O(t(n)^2)$, so the following theorem holds.

Theorem.

$$\mathbf{P} = \mathbf{NP}.$$

10. Conclusion This paper presents the program of polynomial time deterministic multi-tape Turing machine $M_{\exists AcceptPaths}$ that determines if there exist polynomial-length accepting computation paths of polynomial time nondeterministic single-tape Turing machine M_{NP} that decides language A over a finite alphabet. As a result, the equality of classes \mathbf{P} and \mathbf{NP} is proved.

Using the program of machine $M_{\exists AcceptPaths}$, we cannot calculate the count of all the accepting computation paths of machine M_{NP} because we cannot determine when the main loop of the program of machine $M_{\exists AcceptPaths}$ should be stopped. Therefore, if we do not use polynomial $p(n)$, an upper bound of the time complexity of machine M_{NP} , we can only get a lower bound of the upper bound of the count of the accepting computation paths.

However, if we use polynomial $p(n)$ then machine $M_{\exists AcceptPaths}$ would stop at the iteration $p(n)$ and write the count of all the accepting computation paths because there are no accepting computation paths with the length greater than $p(n)$.

References

- [1] D. Du and K. Ko *Theory of Computational Complexity*. New York: John Wiley & Sons, 2000. p.491.
- [2] C. H. Papadimitriou *Computational complexity*. White Plains: Addison-Wesley, 1994. p.523
- [3] S. A. Cook "*The complexity of theorem proving procedures*" in Proc. of the Third Annual ACM Symposium on Theory of Computing, 1971. pp.151-158.
- [4] L. A. Levin "*Universal search problems*" in Problemy Peredaci Informacii 9, pp.115-116, 1973. Translated in problems of Information Transmission 9, pp.265-266.
- [5] S. A. Cook "*The P versus NP Problem*".
Internet: www.claymath.org/millennium/P_vs_NP/pvsnp.pdf
- [6] Cornell University Library. "*Computational Complexity*".
Internet: <http://arxiv.org/list/cs.CC/recent>
- [7] G. J. Woeginger "*The P-versus-NP Page*".
Internet: <http://www.win.tue.nl/~gwoegi/P-versus-NP.htm>